# 3. ПРОЕКТНО – КОНСТРУКТОРСКАЯ ЧАСТЬ

1. **Разработка структуры приложения.**

Данный проект имеет 3 уровневую архитектуру с применением шаблона MVVM

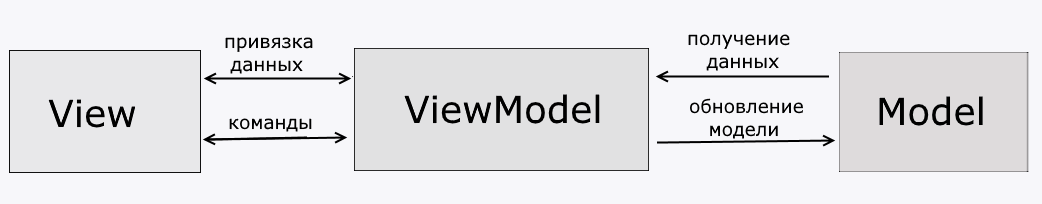


Рисунок 3.1 – Пример работы шаблона MVVM

Для представления принципа работы приложения ниже приведена его UML диаграма вариантов использования

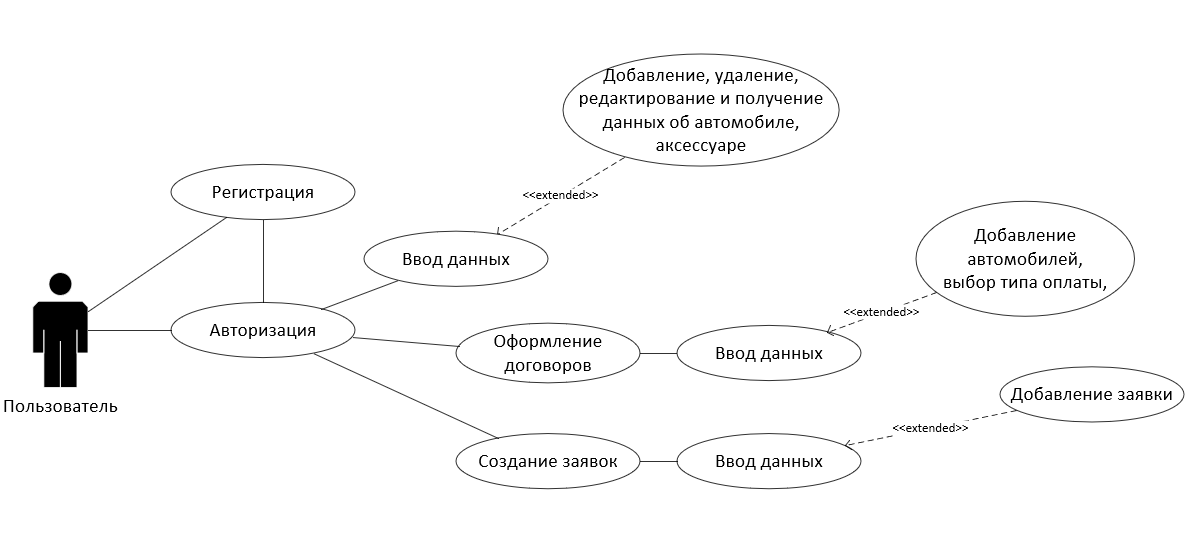


Рисунок 3.2 – Диаграмма использования приложения

В проекте база данных хранить в себе всю информацию о товарах, предоставляемых автосалоном, все действия пользователей приложения и составляемых договоров и заявок. Запросы формируются в слое Model, дальше передаются на обработку в ViewModel и после обработки готовые обработанные данные предоставляться пользователю в слое View. Передача данных также проходит через все слои, пользователь задает параметры или вводит информацию в слое View, дальше данные обрабатываются в ViewModel и передаются в Model, где уже обработанные данные и вносятся или изменяются в базе данных.

Каждой сущности из базы данных соответствует свой класс, для более удобного предоставления информации в слое View. Такое разделение на слои позволяет расширять функционал приложения и постоянно его усовершенствовать. Бизнес-логика выделена в отдельные классы для масштабируемости системы.

Для обеспечения точности и надежности данных были введены ограничения на уровне базы данных. Они включают в себя ограничения уникальности и проверочные ограничения (например, в некоторых таблицах используется ограничение NOT NULL что означает, что нельзя вставить новую запись или обновить запись без добавления значения в это поле). Также были введены ограничения на уровне Бизнес-логики. Например, каждый автомобиль должен иметь свой уникальный VIN номер

Самое главное преимущество MVVM – это технология двусторонней привязки: при изменении модели модель представления будет автоматически обновляться, и представление также автоматически изменится. Это означает хорошую согласованность данных. А также MVVM обеспечивает изолированное модульное тестирование.

1. **Разработка алгоритмов обработки информации.**

3.2.1 Алгоритм подсчета ежемесячного платежа:

Размер аннуитетного ежемесячного платежа высчитывался по формуле

;[[1]](#footnote-1)

Где S — сумма займа, P — 1/100 доля процентной ставки (в месяц), N — срок кредитования (в месяцах).

И также подсчитывалась доля процентов в ежемесячном взносе по формуле

Где S — остаточный объем средств, P — процентная ставка.

От пользователя требовалось указать процентную ставку, срок кредитования в месяцах и первоначальный взнос (если такой присутствовал). Соответственно методу передавались обработанные входные параметры и производился подсчет. В ходе написания алгоритма использовались использовался класс Math из пространства имен System для получения степени числа.

Для тестирования был написан unit-test по результату, которого не было выявлено никаких ошибок

3.2.2 Алгоритм создания Word документа

Для реализации этого алгоритма использовался .NET Framework с использованием библиотеки Microsoft.Office.Interop, а также был выделен отдельный класс ProcessingWord. Для того чтобы реализовать этот алгоритм необходимо было создать заранее шаблон word документа, в котором присутствовали «тэги» для замены их данными.

Конструктор класса ProcessingWord принимал в качестве параметра название шаблона, который располагался в ресурсах проекта и проверял на его наличие. Далее вызывался метод Process который принимал список типа Dictionary<string, string> в котором в качестве ключа хранился «тэг» который был записан в шаблоне, и в качестве значение хранились данные для замены, а также передавалось название папки, куда сохранить созданный word документ. В методе открывается шаблон word документа и далее в цикле перебираются все значения из списка Dictionary<string, string> и ищутся ключи(тэги) уже в самом шаблоне и заменяются на значения из списка. Когда цикл отработал он передает измененный шаблон в функцию SaveAs2 путь куда необходимо сохранить изменений word документ. Функция SaveAs2 копирует все содержимое документа в новый который передаться в качестве параметра, причем после копирования изменённый шаблон не сохраняется. Далее необходимо было предусмотреть все возможные ошибки и избежать ситуации, такой в которой word документ мог не закрыться в связи с окончанием работы или с ошибкой алгоритма. Поэтому в конструкции try-catch-finally в блоке finally закрываются и очищаются все используемые ресурсы. В итоге на выходе мы получаем новый word документ и неизменённый шаблон, который также остается в ресурсах программы.

1. **Логическая схема базы данных**

Проектирование логической структуры БД должно решать задачи выбора наиболее эффективной структуры данных, обеспечения быстрого доступа к данным; исключения дублирования данных, обеспечения целостности данных таким образом, чтобы при изменении одних объектов автоматически происходило соответствующее изменение связанных с ними объектов.

При неправильно спроектированной схеме БД могут возникнуть аномалии модификации данных. Они обусловлены отсутствием средств явного представления типов множественных связей между объектами ПО и неразвитостью средств описания ограничений целостности на уровне модели данных.

Таблица Производители содержит:

* уникальный идентификатор, который определяет брэнд производителя;
* его название;

Таблица Модели содержит:

* номер модели;
* идентификатор, который хранит код производителя для связи таблиц;
* наименование модели;

Таблица Комплектация содержит:

* Код комплектации;
* Идентификатор модели;
* Наименование комплектации;
* Тип привода, 4 вида привода подключаемый, передний, задний или полный;
* Тип топлива, рекомендация производителя для данной комплектации, самые известные типы топлива «АИ-92», «АИ-95», «АИ-98»;

Таблица Автомобиль содержит:

* Код автомобиля для данного автосалона;
* Цвет;
* VIN номер, уникальный номер на кузове транспортного средства который определяет все основные характеристики;
* Год выпуска;
* Номер договора, это поле, может быть, не заполнено если автомобиль продается;
* Код комплектации хранит информацию о комплектации;
* Пробег — это общее расстояние, пройденное транспортным средством;
* Цена;

Таблица Тип аксессуара содержит:

* Код типа аксессуара;
* Наименование типа аксессуара;

Таблица Аксессуар содержит:

* Код аксессуара;
* Код типа аксессуара, который определяет тип аксессуара;
* Наименование аксессуара;
* Цена;

Таблица Договоры содержит:

* Номер договора;
* Код клиента – клиент, с которым заключается договор;
* Код сотрудника– сотрудник, который предоставляет договор;
* Дата – число, месяц и год создания договора;
* Способ оплаты, предоставляемый автосалоном
* Начальный взнос – сумма, которую вносит клиент в качестве оплаты
* Ежемесячный платеж может отсутствовать, но при выдаче рассрочки автосалоном, сумма уплаты в месяц клиентом
* Количество месяцев рассрочки период выдачи рассрочки

Таблица Клиенты содержит:

* Код клиента;
* ФИО – фамилия, имя и отчество клиента;
* Номер телефона – контактный номер телефона для связи с клиентом;

Таблица Сотрудники содержит:

* Код сотрудника;
* ФИО – фамилия, имя и отчество сотрудника;
* Номер телефона – контактный номер телефона для связи с сотрудником;
* Отдел – подразделение, в которое входит сотрудник;
* Должность - служебное место, связанное с исполнением обязанностей;
* Логин уникальный идентификатор пользователя, который используется для входа в систему;
* Пароль – набор символов, которые знает только сотрудник, используется для входа в систему;

Таблица Журнал событий содержит:

* Код события;
* Сотрудник - уникальный идентификатор пользователя;
* Тип – тип события;
* Дата – дата и время события;
* Событие – сообщение системы;

Таблица Заявки содержит:

* Код заявки;
* Тип – тип заказываемого товара;
* Сотрудник - уникальный идентификатор пользователя;
* Наименование заказываемого товара;
* Количество;

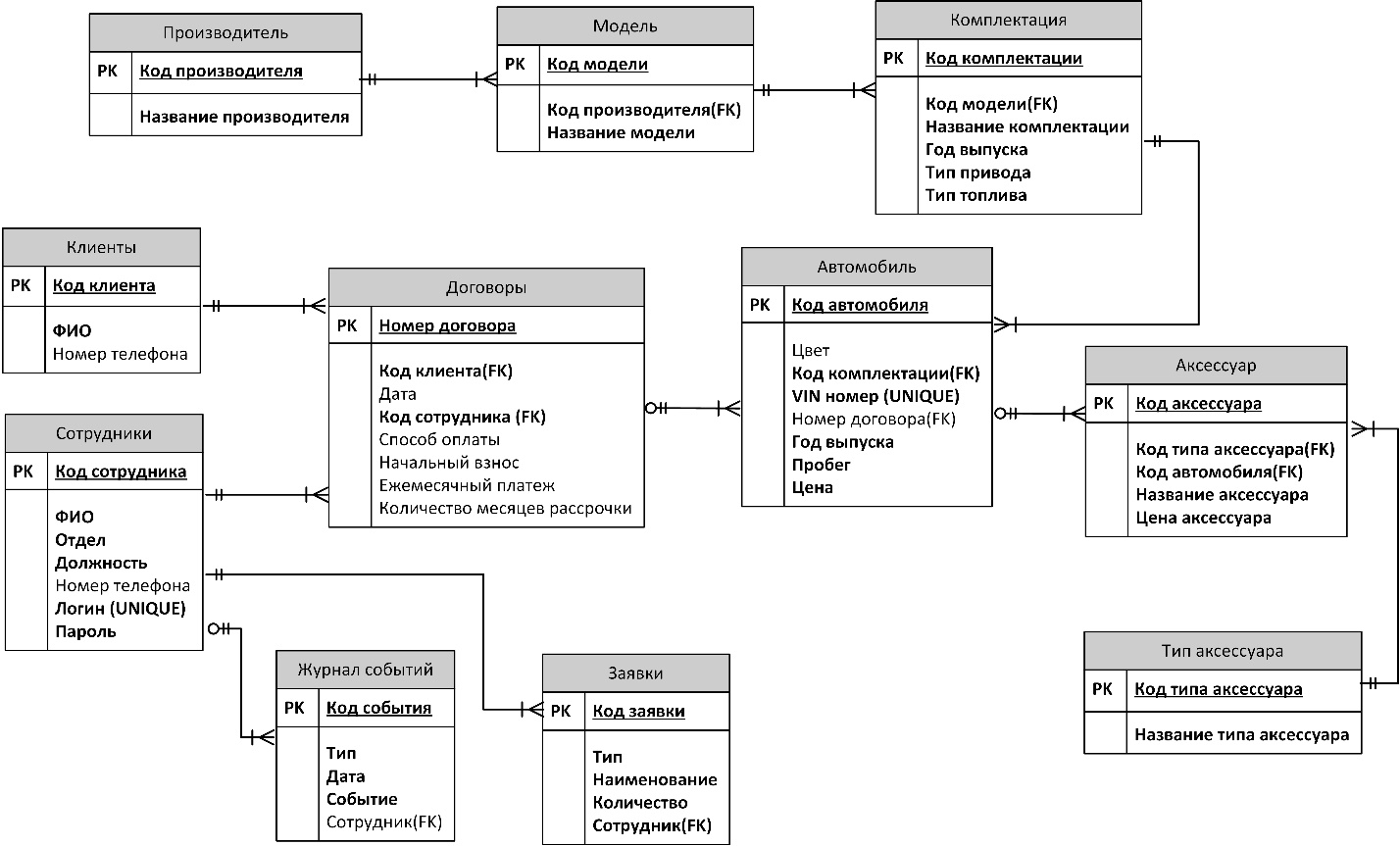


Рисунок 3.3 – Логическая схема базы данных

Таблица находится в 5 нормальной форме, если она удовлетворяет следующим требованиям:

Первая нормальная форма (1NF) предполагает, что сохраняемые данные на пересечении строк и столбцов должны представлять скалярное значение, а таблицы не должны содержать повторяющихся строк.

Вторая нормальная форма (2NF) предполагает, что каждый столбец, не являющийся ключом, должен зависеть от первичного ключа.

Третья нормальная форма (3NF) предполагает, что каждый столбец, не являющийся ключом, должен зависеть только от первичного ключа.

Четвертая нормальная форма (4NF) применяется для устранения многозначных зависимостей (multivalued dependencies) - таких зависимостей, где столбец с первичным ключом имеет связь один-ко-многим со столбцом, который не является ключом. Эта нормальная форма устраняет некорректные отношения многие-ко-многим.

Пятая нормальная форма (5NF) разделяет таблицы на более малые таблицы для устранения избыточности данных. Разбиение идет до тех пор, пока нельзя будет воссоздать оригинальную таблицу путем объединения малых таблиц.

1. **Разработка интерфейса взаимодействия пользователя с системой.**

Для разработки красивого и удобного интерфейса применялась библиотека Material Design. А также были разработаны макеты пользовательского интерфейса:

1. Окно авторизации (Рисунок 3.4).

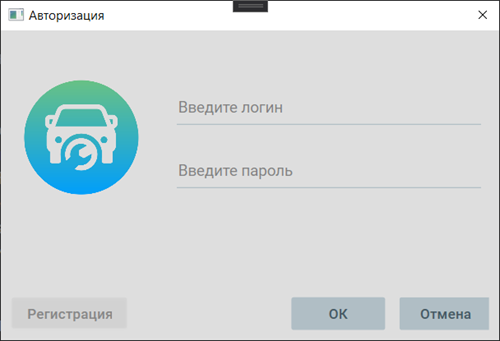


Рисунок 3.4 – Окно авторизации.

1. Окно регистрации (Рисунок 3.5).

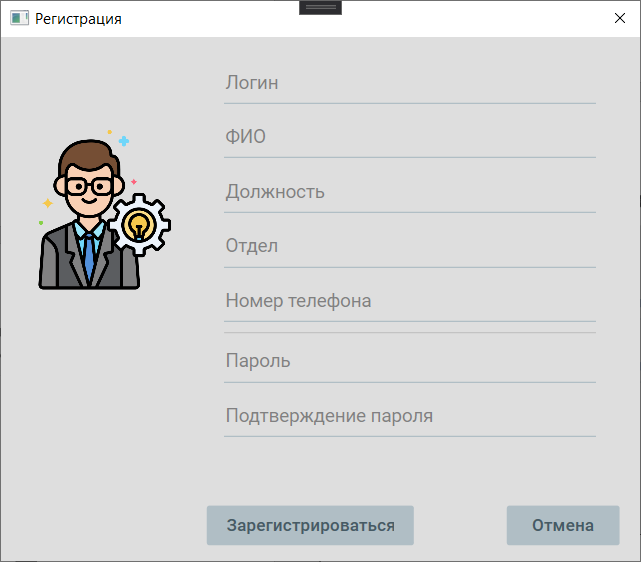


Рисунок 3.5 – Окно регистрации.

1. Страница для работы с автомобилями, добавление, удаление и изменение (Рисунок 3.6).



Рисунок 3.6 – Страница с автомобилями.

1. Окно для добавления производителей, моделей, комплектаций и автомобилей (Рисунок 3.7).

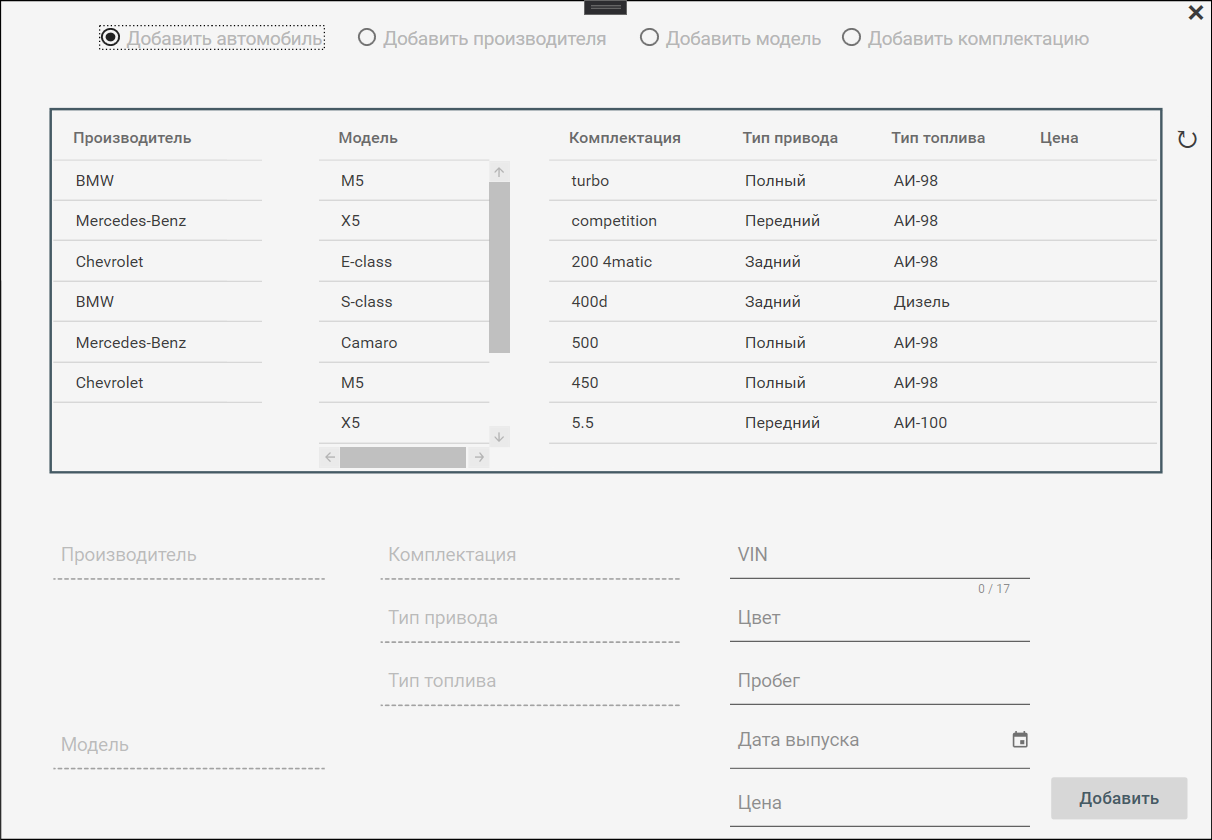


Рисунок 3.7 – Окно добавления автомобилей.

1. Информационное окно об автомобиле, где можно изменить некоторые характеристики (Рисунок 3.8).

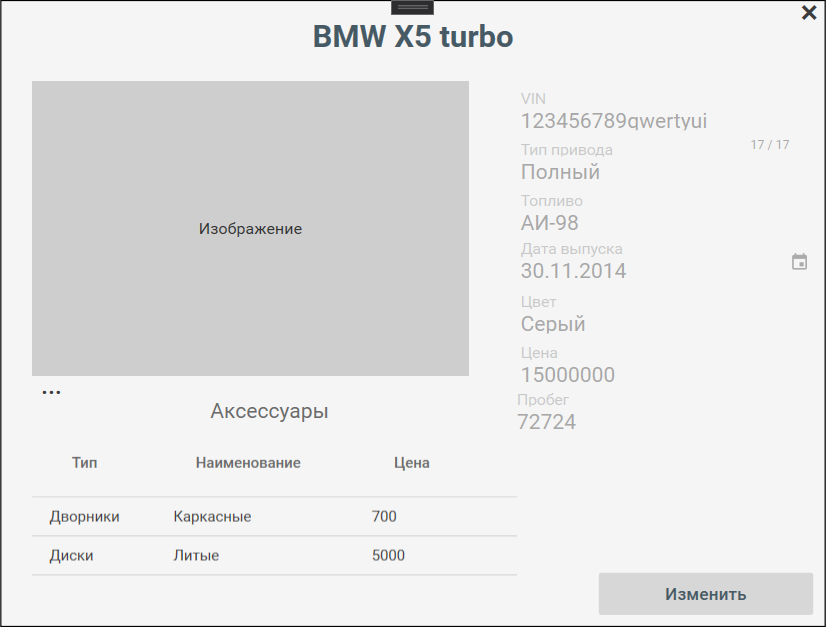


Рисунок 3.8 – Карточка автомобиля.

1. Страница для работы с аксессуарами, добавление, удаление и изменение (Рисунок 3.9).

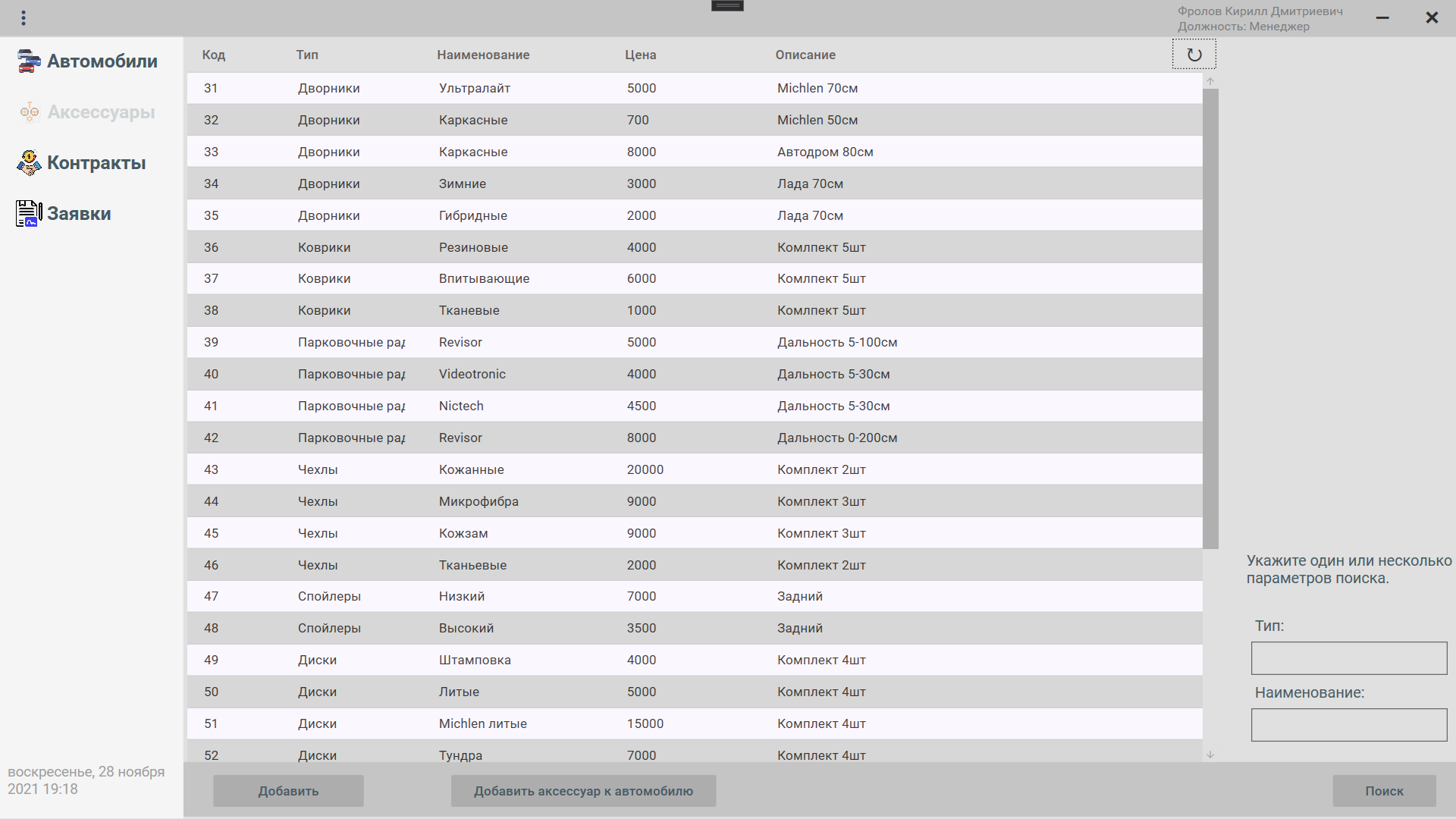


Рисунок 3.9 – Страница с аксессуарами.

1. Окно для добавления типов аксессуаров и аксессуаров (Рисунок 3.10).

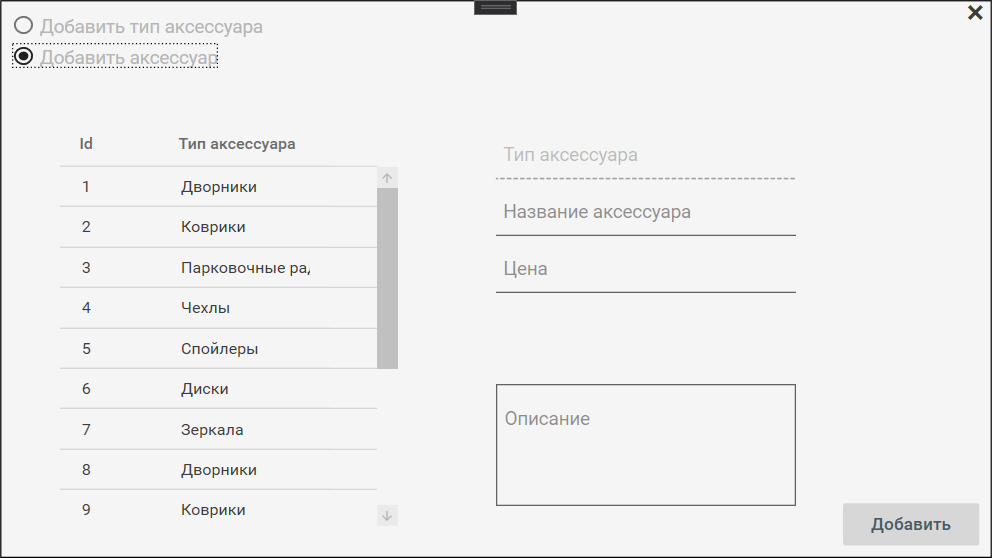


Рисунок 3.10 – Окно добавления аксессуаров.

1. Окно для добавления аксессуаров к автомобилю (Рисунок 3.11).

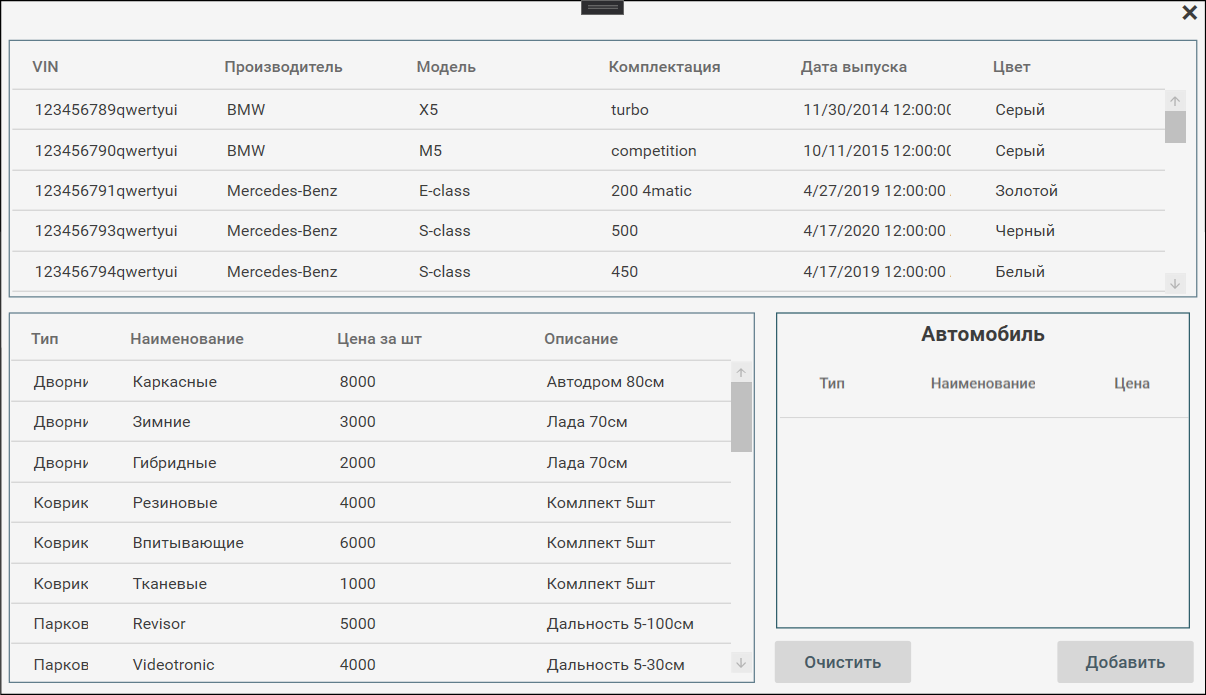


Рисунок 3.11 – Окно добавления аксессуара к автомобилю.

1. Страница для работы с договорами, где можно рассчитать кредит, добавлять автомобили в чек и сформировать документ word на основе введенной и рассчитанной информации (Рисунок 3.12).

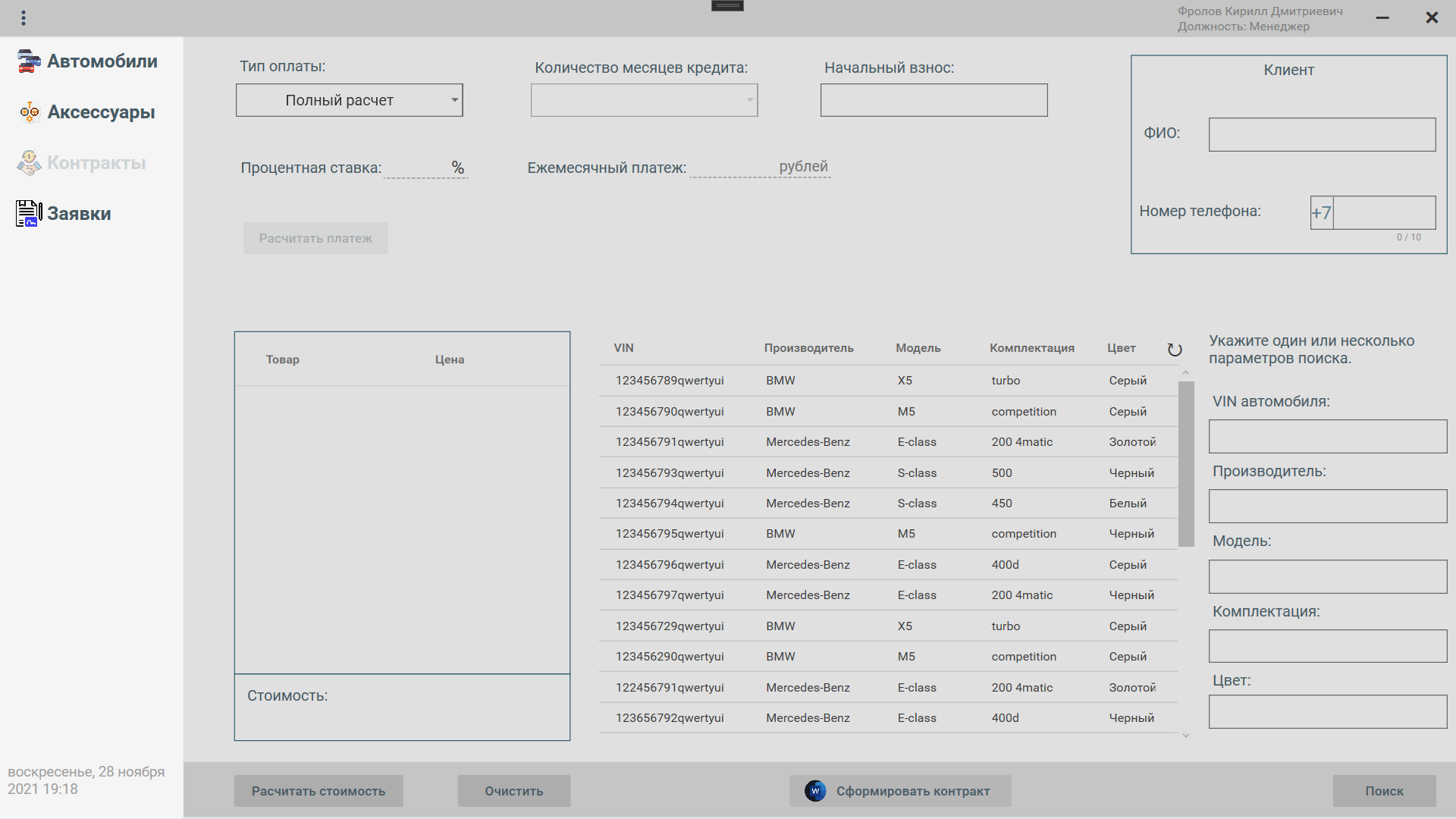


Рисунок 3.12 – Страница для контрактов.

1. Страница для добавления заявок на закупку (Рисунок 3.13).

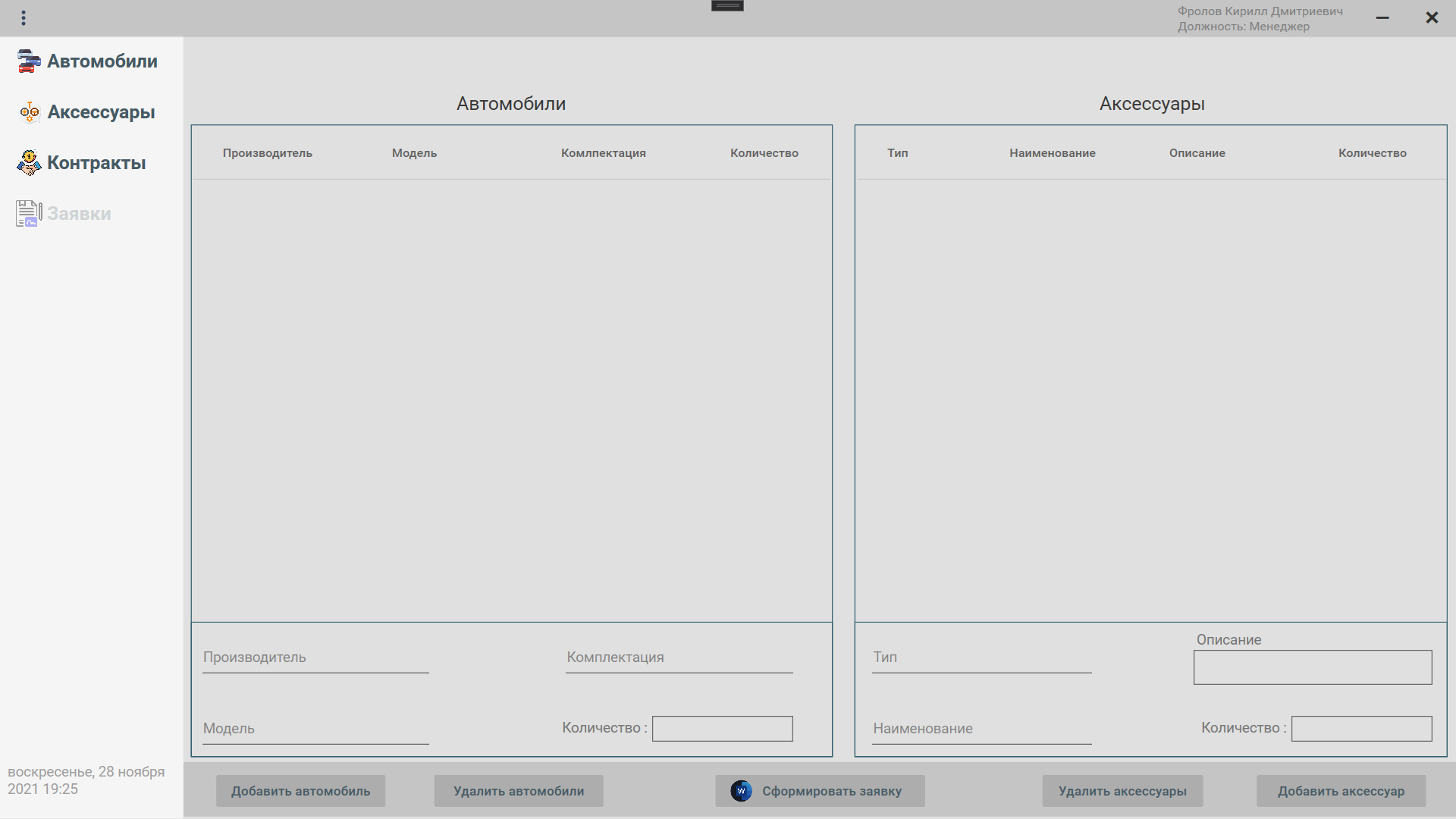


Рисунок 3.13 – Страница с заявками.

1. Страница с информацией о пользователе (личный кабинет) где можно планировать задачи (Рисунок 3.14).

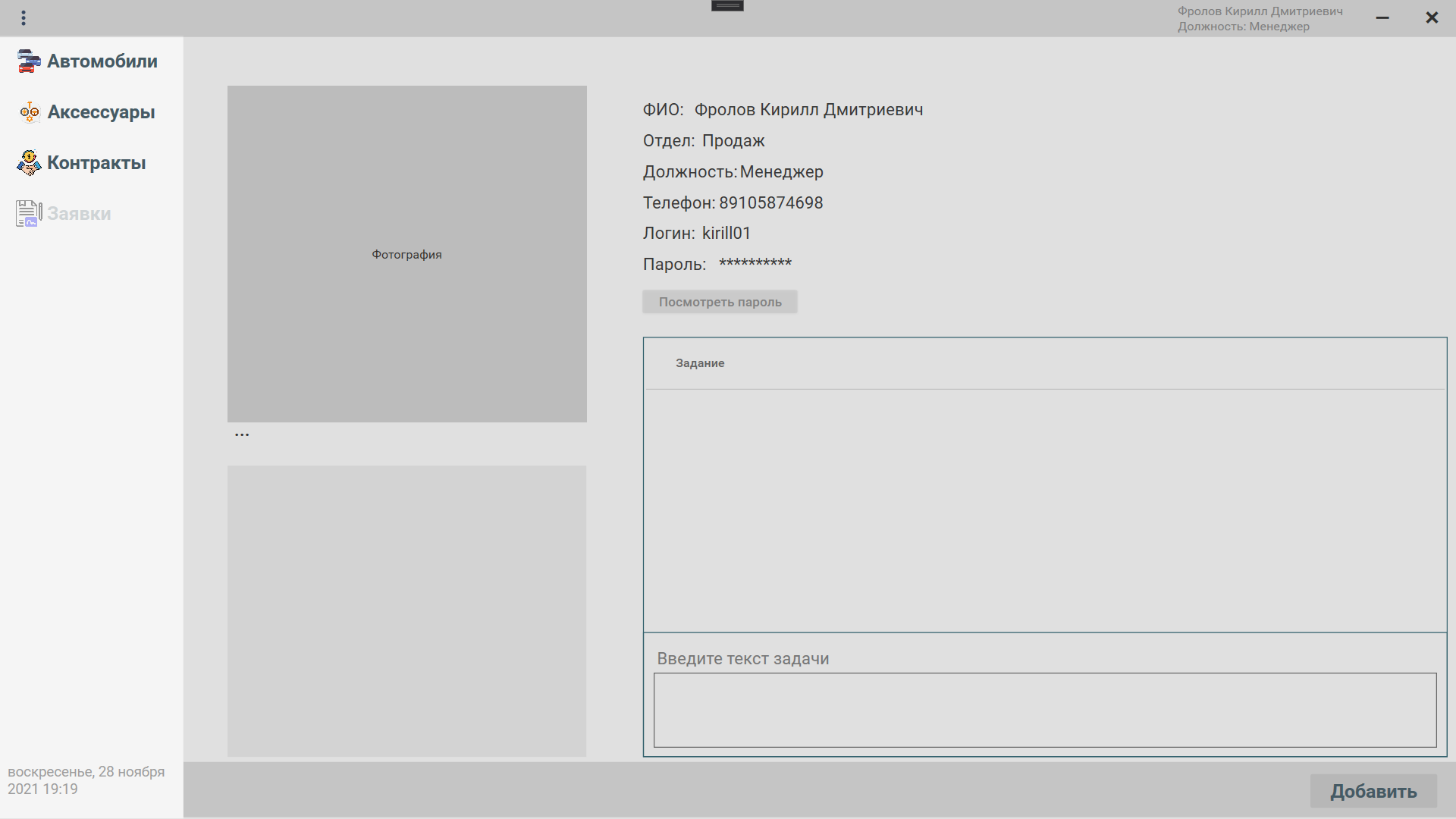
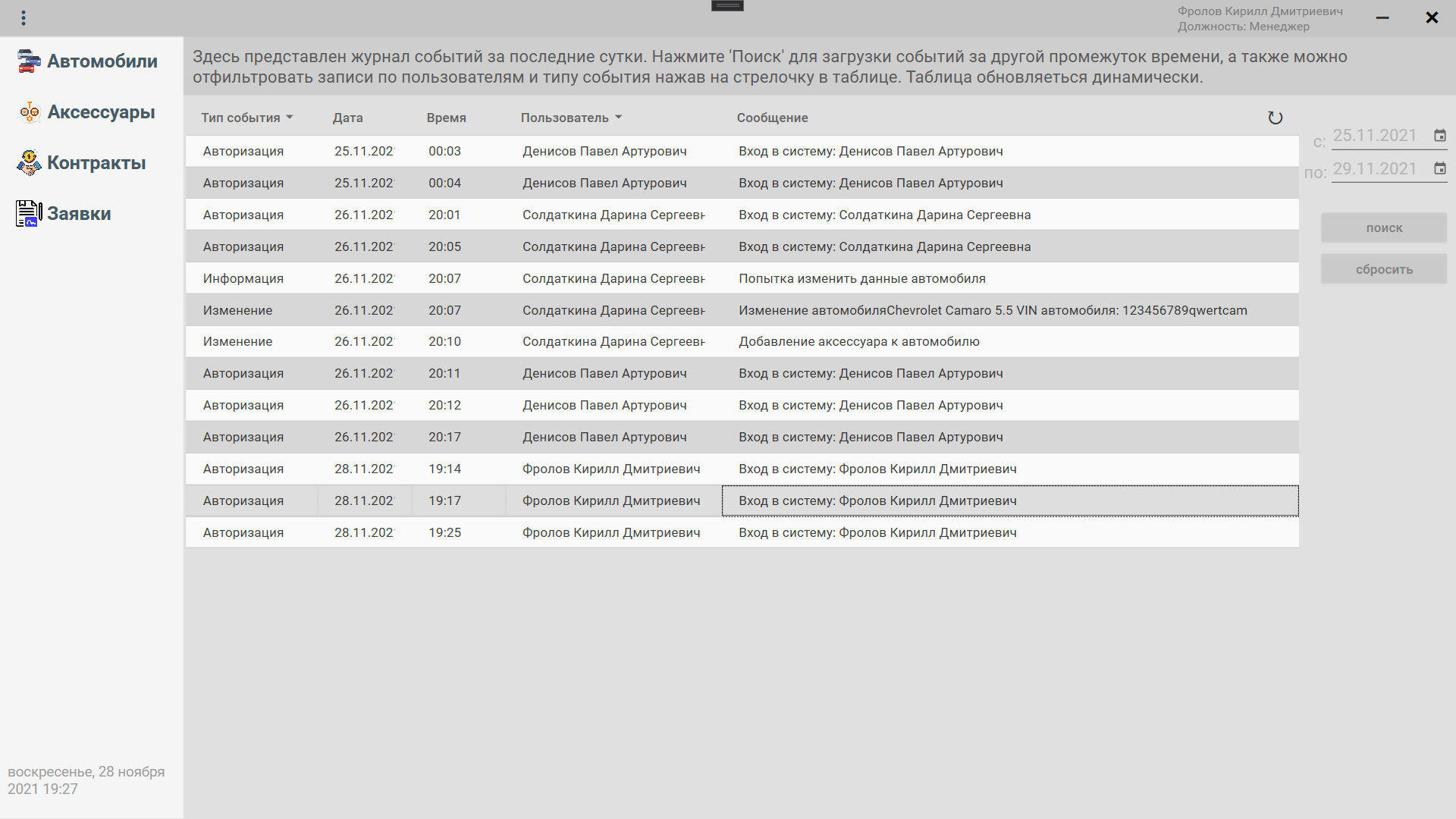


Рисунок 3.14 – Страница личного кабинета.

1. Страница, где можно просматривать последние действия произошедшие в системе (Рисунок 3.15).

  
Рисунок 3.15 – Страница журнала событий.

1. **Разработка архитектуры приложения.**

Было решено создавать настольное приложение, так как оно более эффективно подходит для решения многих задач, связанных с бизнесом, и доступна только в компании. Вэб-приложение не подходило бы для реализации данного проекта, потому что многие компании не часто обновляют системы и соответственно могут быть установлены на ней неподдерживаемые браузеры, которые могут плохо отображать информацию или некорректно работать с данными.

Поэтому был выбран язык программирования C# так как он подробно изучен и хорошо взаимодействует с фреймворком .NET и платформой WPF (Windows Presentation Foundation). При помощи этих инструментов можно создавать клиентские приложения для персональных компьютеров. Так как .NET отличается кроссплатформенностью достаточно будет установить его на операционную систему вместе с приложением и его можно будет использовать. Платформа разработки WPF поддерживает широкий набор компонентов для разработки приложений, включая модель приложения, ресурсы, элементы управления, графику, макет, привязки данных, документы и безопасность. Эта платформа является частью платформы .NET.

В данном проекте используется СУБД PostgreSQL так как она уже хорошо изучена. А также у данной СУБД есть поддерживаемый пакет Nqgsql.EntityFrameworkCore.PostgreSQL для удобной работы с базой данных.

Для связи с базой данных использовался Entity Framework это решение для работы с базами данных, которое используется в программировании на языках семейства .NET. Фреймворк позволяет взаимодействовать с СУБД с помощью сущностей, а не таблиц. Также код с использованием Entity Framework пишется гораздо быстрее. А также использовался язык интегрированных запросов LINQ to Entities для более удобного написания запросов.

В процессе разработки были обнаружены ограничения, связанные с формированием документа word. .Net Core не позволяет использовать в своем решении библиотеку Microsoft.Interop.Office которая предоставляет набор классов и функций для работы с word. Поэтому было принято решение создать отдельный проект на платформе .NET Framework которая позволяет работать с библиотекой Office и использовать этот проект в виде dll библиотеки в основном решении.

1. **Реализация функционирующего приложения.**

Физическая схема данных (Приложение 1). В ходе изучения предметной области было принято решение, ограничить

1. https://www.raiffeisen.ru/wiki/kak-rasschitat-procenty-po-kreditu/ [↑](#footnote-ref-1)